Porting the Spherical Harmonic Transform to Xeon Phi (MIC)

Vincent Boulos, Nathanael Schaeffer

October 23, 2013

Outline

Spherical Harmonic Transform (SHT)

Definition

- Spherical Harmonic of degree ℓ and order m, defined on the sphere :

$$Y_\ell^m(\theta, \phi)$$

- Eigenfunction of the Laplace operator on the sphere :

$$\Delta Y_\ell^m = -\ell(\ell+1) Y_\ell^m$$

Spherical Harmonic Transform (SHT)

Definition

- Spherical Harmonic of degree ℓ and order m, defined on the sphere :

$$Y_\ell^m(\theta, \phi)$$

- Eigenfunction of the Laplace operator on the sphere :

$$\Delta Y_\ell^m = -\ell(\ell+1) Y_\ell^m$$

They form an orthonormal basis

$$f(\theta, \phi) = \sum_{\ell,m} Q_\ell^m \, Y_\ell^m(\theta, \phi)$$

$$Q_\ell^m = \int \int f(\theta, \phi) \, Y_\ell^m(\theta, \phi) \, sin(\theta) d\theta d\phi$$

Application for numerical simulations

Advantages

- Spectral convergence
- Exact derivatives
- Boundary conditions for a magnetic field are straightforward.

Application for numerical simulations

Advantages

- Spectral convergence
- Exact derivatives
- Boundary conditions for a magnetic field are straightforward.

Drawback

As of today, Gauss-Legendre algorithm is the best choice (complexity $\sim \ell_{max}^3$), which means that for high resolution you'll spend most of your time performing SHT !

Implementation

Analysis (forward)

Many Fourier transforms along ϕ, followed by lots of independent (m, θ) series of Legendre polynomial evaluations, which are then summed over θ (quadrature, reduction).

$$f_n^m = \int_0^{2\pi} \int_0^{\pi} f(\theta, \phi) P_n^m(\cos\theta) e^{im\phi} \sin\theta \, d\theta \, d\phi \qquad (1)$$

Synthesis (backward)

Lots of independent (m, θ) series of Legendre polynomial evaluations, summed over ℓ. Followed by many Fourier transforms along ϕ.

$$f(\theta, \phi) = \sum_{m=-N}^{N} \left(\sum_{n=|m|}^{N} f_n^m P_n^m(\cos\theta) \right) e^{im\phi} \qquad (2)$$

SHTns library

Spherical Harmonic Transform on steroids

- Hand-vectorized (SSE2, AVX) using GCC vector extensions.
- Multi-threaded via OpenMP.
- Self-tuning (chooses the best of several variants).
- Compute-bound (on-the-fly generation of Legendre polynomials).
- 80% to 90% of peak performance.

N. Schaeffer, Efficient Spherical Harmonic Transforms aimed at pseudo-spectral numerical simulations, Gcubed, 2013.

First step: compiling with ICC

problems

- CPU: 3 times slower because lack of GCC vector extensions (icc 13)
- MIC: did not compile with `-mmic`

First step: compiling with ICC

problems

- CPU: 3 times slower because lack of GCC vector extensions (icc 13)
- MIC: did not compile with `-mmic`

go ask your question on the intel developper forum !

First step: compiling with ICC

problems

- CPU: 3 times slower because lack of GCC vector extensions (icc 13)
- MIC: did not compile with `-mmic`

go ask your question on the intel developper forum !

solutions

- the intel developper forum is very useful.
- icc 14 (still in beta) supports GCC vector extensions.
- compilation with `-mmic` works after simplification of (non-essential) code.

Second step: FFT on MIC

- MKL available on MIC
- FFT available in MKL, with FFTW interface

problems

- Nowadays, the performance of the FFT is rather memory bound.
- Performance on GPU or MIC is far from peak computing performance.
- On MIC, do not expect more than 100Gflops for the FFT (1000 Gflops is the peak).
- MIC is still better than CuFFT (Nvidia GPUs).
- performance very sensitive to data layout !!

http://software.intel.com/en-us/articles/tuning-the-intel-mkl-dft-functions-performance-on-intel-xeon-phi-copro

https://developer.nvidia.com/cufft

Step 3: Vectorization on MIC

GCC vector extensions (gcc 4+, icc 14+)

- defines vector types holding several scalar values
- use these vector types as if they were scalar (`=, +, -, *, /`)
- dramatically reduces the use of intrinsics (e.g. for reduction)
- allow normal compiler optimization.
- portable

Step 3: Vectorization on MIC

GCC vector extensions (gcc 4+, icc 14+)

- defines vector types holding several scalar values
- use these vector types as if they were scalar (`=, +, -, *, /`)
- dramatically reduces the use of intrinsics (e.g. for reduction)
- allow normal compiler optimization.
- portable

benefit of hand-vectorized code

- MIC has 64 bytes vectors, that is 8 double precision values treated together.
- x2 to x4 overall performance gain compared to auto-vectorized code (including FFT)

http://gcc.gnu.org/onlinedocs/gcc/Vector-Extensions.html

Step 4: Offloading

For large programs and high performance computing, offloading seems mandatory, to allow several nodes to work together (MPI).

- Looks very easy to do: `#pragma offload`
- We have not had much time to work on that
- it is not straightforward to contol allocation and copies on the MIC...
- there are still some compiler bugs
- memory transfer seems slower and less flexible than with OpenCL
- is it possible and easy to do transfer/compute overlap ?

Work still required to make the SHT work well in offload mode

Other remarks

OpenMP thread scheduling

It is possible to control how openmp threads are distributed among cores.
I use explicit `omp_get_thread_num()` which works best (also on cpu).

Other remarks

OpenMP thread scheduling

It is possible to control how openmp threads are distributed among cores. I use explicit `omp_get_thread_num()` which works best (also on cpu).

avoid large private arrays

the analysis algorithm had to be slightly adjusted to get top performance, by using reduction (`_mm512_reduce_add()`) inside the inner loop, while on cpu it is better to postpone it later. This is due to the high number of threads requiring more memory.

Other remarks

OpenMP thread scheduling

It is possible to control how openmp threads are distributed among cores. I use explicit `omp_get_thread_num()` which works best (also on cpu).

avoid large private arrays

the analysis algorithm had to be slightly adjusted to get top performance, by using reduction (`_mm512_reduce_add()`) inside the inner loop, while on cpu it is better to postpone it later. This is due to the high number of threads requiring more memory.

do not necessarily precompute

If a few multiply and add is involved to compute values, do not store them.

Concluding remarks

about Xeon Phi

- It was more work than expected (intel forums are helpful)
- It is definitely less work than porting to OpenCL.
- DGEMM is very fast, do you need it ?
- FFT does not shine (intel, please improve the fft !)
- has still some drawbacks of GPUs (slow memory transfers). Get rid of PCIe ?

Concluding remarks

about Xeon Phi

- It was more work than expected (intel forums are helpful)
- It is definitely less work than porting to OpenCL.
- DGEMM is very fast, do you need it ?
- FFT does not shine (intel, please improve the fft !)
- has still some drawbacks of GPUs (slow memory transfers). Get rid of PCIe ?

about SHT on Xeon Phi

- currently, it is only faster when used in native mode.
- but memory is then limited.
- interesting for desktop users, astrophysics application (very large sizes).
- not sure about heavy simulations on hpc clusters (ivy bridge is better).

Synthesis time comparison

size (ℓ_{max})	cpu 16c	mic	mic offload	tesla m2090	tesla m2090 (2q)
511	1.4	1.5		4.25	2.46
1023	8.9	7.2	16.3	19.3	11.0
2047	62	74.2	117	104.3	68.5
4095	446	296	885	709	547
8191		2050	6850		

Table 1 : Time in milliseconds to complete a spherical harmonic synthesis on various devices and for various sizes. **cpu 16c** is a 16 core 2.7GHz SandyBridge platform. **tesla m2090** with OpenCL (synthesis only) includes the memory transfer (30% to 40%). **2q**: using transfer compute overlap (2 opencl queues). Note that for 511 and 1023, the best times were obtained with "transposed" fft on the mic.

- MIC offload is the slowest (so far, could probably be better)
- MIC native is often fastest (strongly depends on data layout and fft performance !)